

# Rajalakshmi Engineering College

Name: NETHRA CHANDRAGANDHI T  
Email: 240701357@rajalakshmi.edu.in  
Roll no: 240701357  
Phone: 9487531086  
Branch: REC  
Department: CSE - Section 8  
Batch: 2028  
Degree: B.E - CSE

Scan to verify results



## 2024\_28\_III\_OOPS Using Java Lab

### REC\_2028\_OOPS using Java\_Week 8\_CY

Attempt : 1  
Total Mark : 40  
Marks Obtained : 40

#### Section 1 : Coding

##### 1. Problem Statement

Faustus is managing his bank account and wants to create a program to update his account balance based on certain conditions. However, he needs to handle specific scenarios related to invalid inputs and insufficient balances. Faustus wants to update his account balance. He inputs the current balance and the amount to be updated.

The initial account balance should be positive. If Faustus enters a negative initial balance, the program should throw an `InvalidAmountException` with the message "Invalid amount. Please enter a positive initial balance." If the amount to be updated is negative, the program should check if the subtraction results in a negative balance. If so, it should throw an `InsufficientBalanceException` with the message "Insufficient balance." If the amount to be updated is positive, it should be added to the current balance, and the new balance should be printed.

Implement a custom exception, InvalidAmountException, and InsufficientBalanceException, to manage his bank account.

### ***Input Format***

The first line of input consists of a double value 'd', representing the initial account balance.

The second line of input consists of a double value 'd1', representing the amount to be updated.

### ***Output Format***

The output is displayed in the following format:

If the validation passes, print

"Account balance updated successfully! New balance: {new\_balance}"

where {new\_balance} is the updated account balance.

If the initial bank amount is negative it displays

"Error: Invalid amount. Please enter a positive initial balance."

If the updated amount exceeds the initial account balance in withdrawal it displays

"Error: Insufficient balance."

Refer to the sample output for formatting specifications.

### ***Sample Test Case***

Input: 1000  
500

Output: Account balance updated successfully! New balance: 1500.0

### ***Answer***

```
import java.util.Scanner;  
class BalanceUpdater {
```

```

public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);
    try {
        double currentBalance = scanner.nextDouble();
        if (currentBalance < 0) {
            throw new InvalidAmountException("Invalid amount. Please enter a
positive initial balance.");
        }
        double updateAmount = scanner.nextDouble();
        updateBalance(currentBalance, updateAmount);
        System.out.println("Account balance updated successfully! New balance:
" + (currentBalance + updateAmount));
    } catch (InvalidAmountException | InsufficientBalanceException |
java.util.InputMismatchException e) {
        System.out.println("Error: " + e.getMessage());
    } finally {
        scanner.close();
    }
}

```

```

private static void updateBalance(double currentBalance, double
updateAmount)
    throws InvalidAmountException, InsufficientBalanceException {
    if (updateAmount < 0) {
        if (currentBalance + updateAmount < 0) {
            throw new InsufficientBalanceException("Insufficient balance.");
        }
    } else {
        currentBalance += updateAmount;
    }
}

class InvalidAmountException extends Exception {
    public InvalidAmountException(String message) {
        super(message);
    }
}

```

```

class InsufficientBalanceException extends Exception {
    public InsufficientBalanceException(String message) {
        super(message);
    }
}

```

}

**Status :** Correct

**Marks :** 10/10

## 2. Problem Statement

Tim was tasked with creating a user profile system that validates the user's date of birth input. The system should throw a custom exception, `InvalidDateOfBirthException`, if the date is not in the specified format "dd-mm-yyyy" or if it represents an invalid calendar date.

The main method takes user input, validates the date of birth, and prints whether it is valid or not.

### ***Input Format***

The input consists of a string, representing the date of birth of the user.

### ***Output Format***

The output displays one of the following results:

If the entered date of birth is valid according to the specified format, the program prints:

"[Date] is a valid date of birth"

If the entered date of birth is not valid according to the specified format, the program prints:

"Invalid date: [Date]"

Refer to the sample output for formatting specifications.

### ***Sample Test Case***

Input: 01-01-2000

Output: 01-01-2000 is a valid date of birth

### ***Answer***

```

import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.Scanner;
class InvalidDateOfBirthException extends Exception {
    public InvalidDateOfBirthException(String message) {
        super(message);
    }
}
class UserProfileSystem {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        String userInput = "";
        try {
            userInput = scanner.nextLine();
            validateDateOfBirth(userInput);
            System.out.println(userInput + " is a valid date of birth");

        } catch (InvalidDateOfBirthException e) {
            System.out.println(e.getMessage() + ": " + userInput);
        } finally {
            scanner.close();
        }
    }

    private static void validateDateOfBirth(String userInput) throws
InvalidDateOfBirthException {
        SimpleDateFormat dateFormat = new SimpleDateFormat("dd-MM-yyyy");
        dateFormat.setLenient(false);

        try {
            Date dob = dateFormat.parse(userInput);
        } catch (ParseException e) {
            throw new InvalidDateOfBirthException("Invalid date");
        }
    }
}

```

**Status :** Correct

**Marks :** 10/10

### 3. Problem Statement

Theo is trying to update his payment information on a subscription-based streaming service. To proceed, the system requires Theo to provide a valid credit card number consisting of 16 digits. However, Theo wants to make sure that the credit card number he enters meets the specified criteria with proper exception handling.

The credit card number must consist of exactly 16 digits. If the entered credit card number does not meet the specified criteria, the program should throw a custom exception, `InvalidCreditCardException`, and provide Theo with specific error messages: If the length of the credit card number is not 16 digits, the exception message should be: "Invalid credit card number length." If the credit card number contains non-numeric characters, the exception message should be: "Invalid credit card number format."

Implement a custom exception, `InvalidCreditCardException`, to fulfill Theo's requirements and keep his payment information secure.

#### ***Input Format***

The input consists of a string value 's', consisting of the 16-digit credit card number.

#### ***Output Format***

The output is displayed in the following format:

If the entered credit card number is valid, the program should output a success message:

"Payment information updated successfully!"

If the entered credit card has more than 16 digits or less than 16 digits it displays

"Error: Invalid credit card number length."

If the entered 16-digit credit card has non-integers it displays

"Error: Invalid credit card number format."

Refer to the sample output for formatting specifications.

**Sample Test Case**

Input: 1234567890123456

Output: Payment information updated successfully!

**Answer**

```
import java.util.Scanner;
```

```
class InvalidCreditCardException extends Exception {  
    public InvalidCreditCardException(String message) {  
        super(message);  
    }  
}
```

```
class CreditCardValidator {  
    public void validateCreditCardNumber(String creditCardNumber) throws  
InvalidCreditCardException {  
        if (!creditCardNumber.matches("^\\d{16}$")) {  
            if (creditCardNumber.length() != 16) {  
                throw new InvalidCreditCardException("Invalid credit card number  
length.");  
            } else {  
                throw new InvalidCreditCardException("Invalid credit card number  
format.");  
            }  
        }  
    }  
}
```

```
class CreditCardUpdater {  
    private CreditCardValidator validator = new CreditCardValidator();  
  
    public void updateCreditCard() {  
        Scanner scanner = new Scanner(System.in);  
        try {  
            String creditCardNumber = scanner.nextLine();  
  
            validator.validateCreditCardNumber(creditCardNumber);  
        }  
    }  
}
```

```

        System.out.println("Payment information updated successfully!");
    } catch (InvalidCreditCardException | java.util.InputMismatchException e) {
        System.out.println("Error: " + e.getMessage());
    } finally {
        scanner.close();
    }
}
}
}
public class Main {
    public static void main(String[] args) {
        CreditCardUpdater updater = new CreditCardUpdater();
        updater.updateCreditCard();
    }
}
}

```

**Status :** Correct

**Marks :** 10/10

#### 4. Problem Statement

Alice is designing a program that requires users to enter positive numbers. She wants to implement a solution that validates whether the entered number is positive. In case the input is not a positive number, she wants to throw a custom exception.

The number should be a positive integer. If this condition is violated, the program should throw a custom exception: `InvalidPositiveNumberException` with the message "Invalid input. Please enter a positive integer."

Implement a custom exception, `InvalidPositiveNumberException`, to handle cases where the entered number does not meet the specified criteria.

##### **Input Format**

The input consists of an integer value 'n', representing the entered number.

##### **Output Format**

The output is displayed in the following format:

If the validation passes, print



"Number {number} is positive."

The {number} represents the entered positive integer.

If the entered number is negative then it displays

"Error: Invalid input. Please enter a positive integer."

Refer to the sample output for formatting specifications.

### **Sample Test Case**

Input: 100

Output: Number 100 is positive.

### **Answer**

```
import java.util.Scanner;
class PositiveNumberValidator {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        try {
            int number = scanner.nextInt();
            validatePositiveNumber(number);
            System.out.println("Number " + number + " is positive.");
        } catch (InvalidPositiveNumberException | java.util.InputMismatchException
e) {
            System.out.println("Error: " + e.getMessage());
        } finally {
            scanner.close();
        }
    }

    private static void validatePositiveNumber(int number) throws
InvalidPositiveNumberException {
        if (number <= 0) {
            throw new InvalidPositiveNumberException("Invalid input. Please enter a
positive integer.");
        }
    }
}
```

```
}  
class InvalidPositiveNumberException extends Exception {  
    public InvalidPositiveNumberException(String message) {  
        super(message);  
    }  
}
```

**Status :** Correct

**Marks :** 10/10